Locks & Isolation Levels in SQL Server

INTRODUCTION

Maintaining data integrity and durability are among the basic principles of any RDBMS. As one of the well-known RDBMSs, SQL Server is no exception to this rule and has benefited from the concept of Transactions. Based on the isolation feature of all transactions, the changes made in each transaction is hidden and isolated from other transactions until its commitment. In other words, the scope for each transaction is like its privacy.

Unlike a database on your personal laptop, unfortunately or maybe fortunately in a real operating environment, you are not the only person using the database and there might be hundreds or thousands of other people working with that database and they might even request to use the same resources. In such cases, Transaction Isolation will ensure that the result of the simultaneous execution of these requests / transactions will be the same as their serial execution, which means that the simultaneous execution of transactions will not have a negative impact on each other or will not lead to legal violations. Such a result will be achieved by adjusting and controlling the Transaction Isolation Levels.

Transaction Isolation Level determines the level of rights other transactions have to access the available resources or the resources modified by the current transaction. This behavior is supported by the way resources are locked and the period of keeping them locked.

When writing SQL Server queries or designing packages in SQL Server Integration Service (SSIS), in addition to creating and determining the Transaction scope, the type of Transaction Isolation Level used by that transaction can also be determined.

Before explaining the types of Isolation Levels, you need to be familiar with the concept of Lock and its types. Although the explanation of this category does not fit into this concept, here are brief explanation of some of the most popular locks.

- **Lock:** SQL Server uses the lock concept to support and implement the Isolation transaction feature. Locks can be retrieved and stored on a variety of sources, including records, pages, partitions, tables, and databases. You need to keep in mind that the granularity of the locks will definitely affect decreasing or increasing of the concurrency.
- **Exclusive (X) Lock:** This type of lock is obtained by any command that intends to change the data of a source, or so-called Writers. This lock is the most incompatible type of lock, which means that if a transaction wants to get a resource that already has an X lock on it, it will be blocked until the transaction is completed (Commit or Rollback) and has to wait. Obviously, only one transaction will be able to get this type of lock on a resource at any given time.
  The important thing to keep in mind is that setting Transaction Isolation Levels for these types of transactions will have no effect on duration of this type of locks, and locks will always remain on the resources until the transaction is completed.
- **Shared (S) Lock:** This type of lock is obtained by Read commands on resources and is compatible with other locks of the same type. Therefore, the same source can be shared between multiple Readers.
- **Update (U) Lock:** This type of lock is also obtained by Writers. SQL Server uses this type of lock to evaluate whether or not this line will need to be updated before updating a source and obtaining a lock (X) on it. After the lock (U) is obtained on a line, the SQL Server will check the blind condition. If that line needs to be updated, it will convert the U lock to X, otherwise it will release this lock.
  Compatibility of this type of lock with lock (S) prevents unnecessary or premature blocking of Writers and Readers.

Types of Isolation Level

Types of Transaction Isolation Levels can be classified into two general categories: Pessimistic Isolation Levels and Optimistic Isolation Levels.

Each of these methods has its own advantages and disadvantages, and each will solve some of the problems of synchronicity.

It is emphasized that determining the Isolation Level will not affect the locks obtained by the data modification commands. When a transaction changes information, regardless of the Transaction Isolation Level assigned to it, it will always keep the exclusive lock (X) obtained until the end of the transaction.

Pessimistic Isolation Level

In pessimistic methods, it is assumed that several transactions will most likely be accessed by a common source at the same time with the intention to change. The basis of such methods is to lock and block other simultaneous transactions that require incompatible lock requests.

- **Read Uncommitted**
  When you use this Isolation Level, you will not create any Shared Lock for the resources you need. In this case, the current transaction will be able to enter the privacy of other transactions and read their uncommitted data, because without using the lock, other transactions will not be aware of the existence of this transaction, and as a result, there won't be a blocking due to existence of incompatible locks. With this Isolation Level you will accept that you are aware of all the problems of synchronization, including Dirty Reads.
- **Read Committed**
  Shared Lock will be created on the required resources when using this type of Isolation Level. As a result, transactions with this type of Isolation Level can block other transactions with incompatible locks or they are blocked themselves by these types of transactions. The important thing about this type of Isolation Level is that the locks obtained on the resources at each level of granularity will be released after the completion of the reading operation. For example, a Row Lock will be released after processing that line, or a Page Lock will be released after processing that page.
  Obviously, in this Isolation Level, the Dirty Read problem has been solved before, but there are still other problems that we will introduce later in the session.

- **Repeatable Read**
  In this type of Isolation Level, like the Read Committed method, Shared Lock is used on the available resources, but unlike the previous method, the obtained locks will remain until the end of the transaction. Although this will cause long-term blocking of other incompatible transactions, the majority of synchronous problems will be solved.
- **Serializable**
  In this method, which is the strictest type of Isolation Level, not only will the Shared Lock obtained on the resources remain until the end of the transaction, but the use of SQL Server from a type of lock called Range Lock will cause a range of index key to be protected instead of single lines. In this case, if another transaction wants to insert a new line whose key value is in the mentioned range (protected), it will be blocked. In this type of Isolation Level, all synchronization problems are solved. However, the Concurrency rate is at its lowest possible level.

It should be noted that the default Isolation Level of transactions in SSIS, unlike SQL Server, is Serializable.

The most common concurrent problems in different Isolation Levels

- **Dirty Read:** This problem will occur when a transaction accesses and reads the non-Committed data of another transaction.
- **Non-Repeatable Read:** This problem will occur when successive attempts to read the same data within a transaction lead to different results. The cause of this problem can be considered the release of locks obtained by the commands inside a transaction before completion of that transaction.
- **Phantom Read:** This problem occurs when successive attempts to read the same data within a transaction lead to the return of new records that the transaction did not read in the previous reading.(Can read inserted records from other transactions before commit)-(Ghost records)

	Dirty reads	Non-Repeatable Reads	Phantom Records
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READS	No	No	Yes
SERIALIZABLE	No	No	No

Optimistic Isolation Level

Optimistic methods have been proposed with the aim of solving blocking problems as well as solving data consistency problems. In these methods, it is assumed that, most likely, a shared resource will be changed by simultaneous transactions. This type of method is based on Row Versioning along with locking.

- **Read Committed Snapshot**
  In fact, this type of Isolation Level is a setting at database level, and it can't be considered as an independent Isolation Level. It will not affect the behavior of writers. When this setting is enabled on a database, any data changes will cause the previous version to be stored in a space of TempDB database called the Version Store. If a Read command encounters an uncommitted version of another transaction when accessing data, it uses the data before the start of that command (Statement) instead of blocking the last committed version of that data. Statements (commits) are not available by this command.
  Although this method solves the problem of blocking readers and writers, writers will still block each other due to the use of U and X locks.
- **Snapshot**
  Before using this type of Isolation Level, you need to enable this feature on the database you want. One of the main differences between this method and previous methods is that this method creates a data consistency at the Transaction level (Transaction-Level Data consistency) without blocking and commands within a transaction only have access to the last version committed from the information before the initiation of transaction. Writers' performance in this Isolation Level will be different. When a writer encounters a modified line, he or she will refer to the previous version of that line in the Version Store to check the update condition, without being blocked due to the incompatibility of the U and X locks. Only when it needs to be updated will it put an X lock on that line, in which case it will end by be one of these situations: Successful update, being blocked by another X lock or receiving an error due to encountering a new version of that line.

Create a New User in SQL Server

There are two ways to create a new User or to grant user permissions:

- Using Microsoft SQL Server Management Studio
- Using T-SQL

```
USE TRAINEES

GO

    CREATE USER 'John' FOR LOGIN 'sa'

GO
```

Assign Permissions to User in SQL Server

You can grant/revoke any or a combination of the following types of permissions:

Select: Grants user the ability to perform Select operations on the table.
Insert: Grants user the ability to perform the insert operations on the table.
Update: Grants user the ability to perform the update operations on the table.
Delete: Grants user the ability to perform the delete operations on the table.
Alter: Grants user permission to alter the table definitions.
References: References permission is needed to create a Foreign key constraint on a table. It is also needed to create a Function or View WITH SCHEMABINDING clause that references that object
Control: Grants SELECT, INSERT, UPDATE, DELETE, and REFERENCES permission to the User on the table.

```
USE TRAINEES;
  GRANT SELECT, INSERT, UPDATE, DELETE ON Employee TO John;
USE TRAINEES;
  GRANT SELECT ON Employee TO public;
USE TRAINEES;
  REVOKE DELETE ON Employee FROM John;
```

ROLE CREATION

```
CREATE SCHEMA Sales;
GO
CREATE USER Joe without login;
GO
CREATE ROLE Vendors;
GO
```

```
ALTER ROLE Vendors ADD MEMBER Joe;
GO
GRANT SELECT ON SCHEMA :: Sales TO Vendors;
GO
REVOKE SELECT ON SCHEMA :: Sales TO Vendors;
GO
```

END OF MS SQL SERVER